DEVELOPING A SUCCESSFUL METRICS PROGRAM

LINDA ROSENBERG, Ph.D.

Unisys /GSFC

Bldg 6 Code 300.1

Greenebelt, MD 20771 USA

Tel: 301-286-0087

Fax: 301-286-0322

LAWRENCE HYATT

GSFC NASA

Bldg. 6 Code 302

Greenbelt. MD 20771 USA

Tel: 301-286-7475

Fax: 301-286-1701

ABSTRACT

This paper discusses how affordable metric programs can be applied to help project managers and developers evaluate the quality of their project's products, to help them evaluate and track project risks, and to assist in process and product improvement. This paper first discusses the foundation of a software quality metrics program based on the evaluation of product quality and project process effectivity. The quality and effectivity measures are then related used to indicate project risk and to improve processes and future products. A basic metrics program is then outlined that is applicable to any software development project. Data from NASA GSFC projects are used to demonstrate the metrics and their interpretation in terms of risk. Suggestions for starting a metrics program are discussed, based on the application of the Goal/Question/Metric Paradigm. Costs and benefits of a metrics program are included.

This paper supplies both project managers and software developers with techniques to initiate a metrics program that yields timely, relevant, usable information at minimal cost.

KEY WORDS - Metrics, Software, Quality, Risks

1. INTRODUCTION

The Software Assurance Technology Center (SATC) was established at NASA's Goddard Space Flight Center (GSFC) as a center of excellence in software assurance, dedicated to making measurable improvement in the quality and reliability of software developed for GSFC and NASA.

The objectives of the software metrics program of the SATC encompass three areas: Quality Assessment; Risk Management and Control; and Product and Process Improvement. To accomplish these objectives, the SATC works with project managers and developers (both NASA and contractor) to collect measurements and develop metrics that assist them in evaluating the quality of the products (requirement documents, design documents, code, test plans, etc.) and the effectivity of their processes. Using this information the risks to their projects is assessed.

Developing a metrics program is not easy. It has many possible pit-falls that can lead to the ruin of the metrics program itself and possibly the project if incorrectly applied. [2] This paper first discusses the foundation of a software quality metrics program based on the evaluation of product quality and project process effectivity. The quality and effectivity measures are then used to indicate project risk and to improve processes and future products. A basic metrics program is then outlined that is applicable to any software development project. Data from NASA GSFC projects is used to demonstrate the metrics and their interpretation in terms of risk. Suggestions for starting a metrics program are discussed, based on the application of the Goal/Question/Metric Paradigm. Costs and benefits of a metrics program are included.

2. SOFTWARE QUALITY METRICS PROGRAM

The SATC has developed a Software Quality Metrics Program and associated metrics that supports risk management and quality assessment of the processes and products of software development projects. The SATC program, shown in Figure 1, contains dynamic elements, so that projections in time can be made to effect the direction of the project. It takes into consideration project goals and milestones, so that it can be tailored for the particular project. It assists in determining risks, so that management action can be taken. The combination of these three elements makes the SATC program unique.

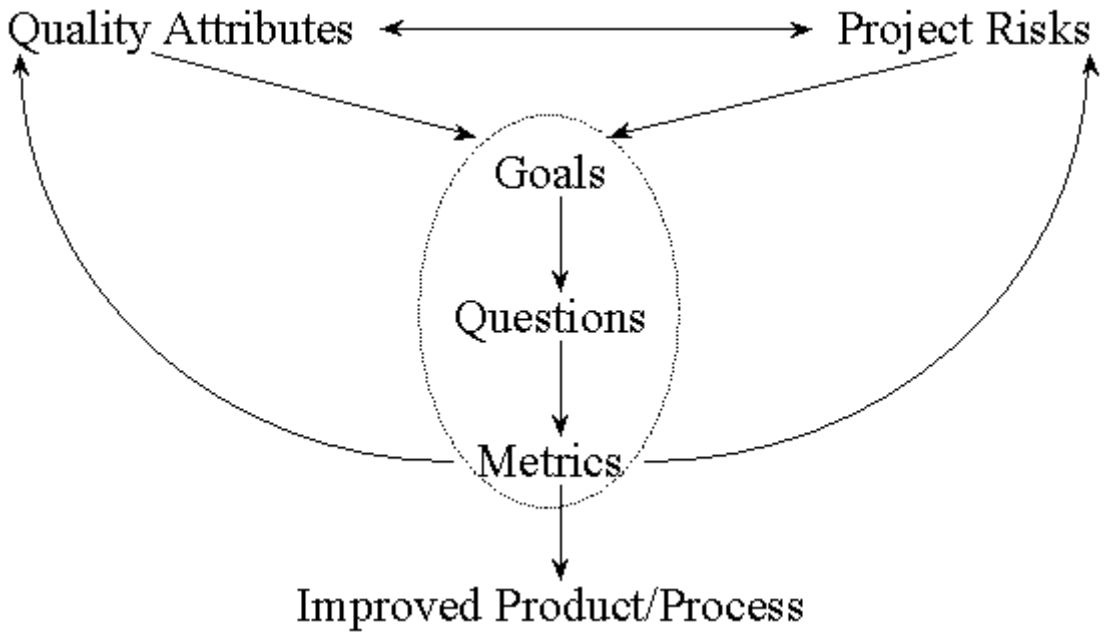

Figure 1: SATC Model for Software Metrics Program

The SATC model for metrics programs defines a set of goals that span the life cycle. The goals are then related to software product and process attributes and a set of metrics is chosen or developed that measures the attributes. [4] The four goals are:

1. Requirements Quality

2. Product Quality
3. Testing Effectivity
4. Process Implementation Effectivity

2.1 GOAL 1: REQUIREMENTS QUALITY

The objectives for this first goal are complete, unambiguous, and understandable requirements, the stabilization of requirements as quickly as possible, and the traceability of all requirements from their source to the software requirements and then through design to implementation and test. The associated attributes are:

- **Ambiguity** - Potential multiple meanings.
- **Completeness** - Items left to be specified.
- **Requirement Volatility** - The rate and the life cycle phase when requirement changes are made.
- **Traceability** - The traceability of the requirements upward and downward.

2.1.1 REQUIREMENT ATTRIBUTES AND METRICS

Ambiguous requirements are those that may have multiple meanings or those that seem to leave to the developer the decision whether or not to implement a requirement. There are two metrics used to evaluate the ambiguity of the requirements document: the number of weak phases (e.g. adequate, as appropriate, normal ,timely normal, timely) and the number of optional phrases (e.g. can, may, optionally). Ambiguity is computed as the sum of the count of weak phrases and the count of optional phrases.

If the requirements document is complete, it will have all of the requirements specified in adequate detail to allow design and implementation to proceed. The metrics used to evaluate completeness are a count of the TBA (to be added) and TBD (to be determined) and similar acronyms used in the document. These acronyms are those most often used to indicate that a portion of the requirements must be supplied at some future time. [6]

The SATC has developed a tool that uses a text file copy of the requirements document as input and returns a report containing counts of structures, including the number of ambiguous terms, incomplete terms (TBD, TDA, etc.), and counts of the number of requirements. The Automated Requirement Measurement tool (ARM) is available from the SATC web site http://satc.gsfc.nasa.gov/SATC/TOOLS/ARM/armpage.html.

Volatility measures the rate of change of requirements. The impact of changes to the requirements increases as one gets closer to the time when the software is to be released. The metrics for volatility are then the percentage of requirements changed in a given time period, computed as number of requirements in the changes to the document divided by the base count of requirements.

Software requirements must be traceable to system requirements to assure that the software when

developed will work properly in the system setting. The software requirements must also be traceable to the implementing design and code, to ensure that they are in fact implemented. They must be traceable to tests to ensure that they have been validated and verified. There are two metrics for traceability, the number of requirements not traced to higher level requirements and the number not traced to code and tests. [3]

2.1.2 REQUIREMENTS RISK

It is generally accepted that poorly written, rapidly changing requirements are a principal source of project risk (and indeed, of project failure). To reduce risk, specific requirements that are ambiguous or that contain TBAs or TBDs can be identified and revised.

Volatility is an important factor in risk, and extensive measures are often taken to reduce its impact. The later in the life cycle changes are made to requirements, the more resources needed to implement them. The earlier in the life cycle the requirements stabilize, the less the risk.

Overall assessment of requirements risk has to blend the impact of both types of the requirements metrics. It should be noted that requirements risk must be measured throughout the life cycle.

2.2 GOAL 2: PRODUCT QUALITY

An important objective of a software development project is to develop code and documentation that will meet the project's "ility" requirements. The specific attributes for product quality are:

- **Structure/Architecture** - The evaluation of the constructs within a module to identify possible error-prone modules and to indicate potential problems in usability and maintainability.
- **Reuse** - The suitability of the software for reuse in a different context or application.
- **Maintainability** - The suitability of the software for ease of locating and fixing a fault in the program.
- **Documentation** - The adequacy of internal code documentation and external documentation.

2.2.1 PRODUCT QUALITY ATTRIBUTES AND METRICS

The attributes of Structure/Architecture, Reusability and Maintainability use the same metrics, but with different emphasis. The metrics are complexity, size, and the correlation of complexity with size.

It is generally accepted that more complex modules are more difficult to understand and have a higher probability of defects than less complex modules. Thus complexity has a direct impact on overall quality and specifically on maintainability. Projects developing code intended for reuse should be even more concerned with complexity since it may later be found to be more expedient to rewrite highly complex modules than to reuse them. While there are many different types of complexity measurements, the one used by the SATC is logical (Cyclomatic) complexity, which is computed as the number of linearly independent test paths.

Size is one of the oldest and most common forms of software measurement. Size of modules is itself a quality indicator. Size can be measured by: total lines of code, counting all lines; non-comment non-blank which decreases total lines by the number of blanks and comments; and executable statements as defined by a language dependent delimiter.

While both size and complexity are useful metrics, the correlation of the two produces even more information. The SATC has found that much of the code produced at GSFC has a linear relationship between complexity and size, such that the complexity of a "normal" module can be predicted by its size. Those modules above the linear relationship - those that are more than normally complex - are the ones whose quality is suspect. This is demonstrated in Figure 2. [5]

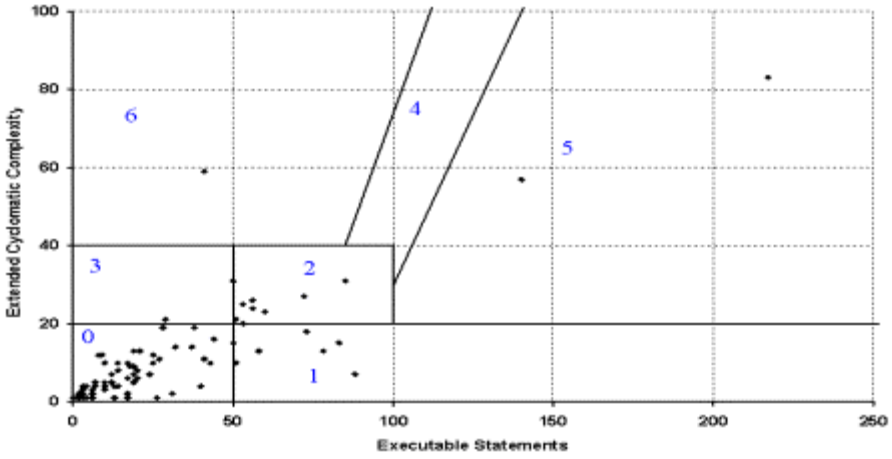

Figure 2: Size to Complexity

Documentation is the description of the content of the code. The metric for this attribute is the percentage of comments within the code. Comment percentage is calculated by dividing the number of comments by the total lines of code less the blank lines.

2.2.2 PRODUCT QUALITY RISK

While poor quality of any product is a risk to the specific objectives of the project, some of the best measures of risk come from correlations of the base metrics. For example, the SATC uses a scatter plot of the complexity versus the size of all of the code modules in a program or segment of the system software as shown in Figure 2. The scatter plot identifies the risks to correctness and, especially, maintainability and reusability. Risks are assigned to areas in the scatter plot based on the SATC's experience.

The risk determined by the complexity - size correlation can be combined with the commenting percentage for high risk modules to identify code that should be improved.

2.3 GOAL 3: TESTING EFFECTIVITY

The objectives for effective testing is to locate and repair faults in the software, to identify error-prone software, and to complete testing on schedule with sufficient faults found and repaired that the software will operate "well enough" when it is put into operation. The attributes measured

are:

- **Correctness** - Freedom from Faults
- **Error Concentrations** - Localization of errors and high criticality errors in specific software segments
- **Comprehensiveness** - Amount of coverage

2.3.1 TESTING EFFECTIVITY ATTRIBUTES AND METRICS

Correctness is defined as the extent the code fulfills specifications. One implication is that the software must be error free. Measurements of errors are only recorded during system level tests. Comprehensiveness is the evaluation of the coverage of the requirements through testing. It is also important to monitor the location of the errors within the code since areas of high error concentration may have multiple design changes, and potentially be the source of additional errors.

The SATC has developed a model that uses the typical cumulative error curve equation and does a curve fit with the error data from testing. After the projections have settled down (about a third of the way through the test process), the fitted curve can be used to predict the total number of errors that will be found and the schedule resource needed to find them. We have also have had some success in predicting the total number of errors by criticality category.

Correctness is reported as the number of errors found and, after the model has settled down, the expected total number of errors that will be found.

2.3.2 TESTING EFFECTIVITY RISK

The correctness measurements lend themselves to a number of risk projections. If the project has quantified the percentage of the errors that are to be found before the software is accepted, the model can be used to project when that will happen. If the time is significantly after the time at which the schedule requires the software to be available, then risk is high.

Another risk determination can be done by looking at the code modules or sub-systems in which the errors occur. Segments of code with more than the average number of errors may well need another look to see if re-engineering should be considered. Also, the time to fix errors should be tracked. If this is increasing, then there is a risk to correctness and to schedule.

Finally, the location of faults that caused high criticality errors should be tracked. A concentration of them in a segment of code indicates a risk that the requirements are not well understood, or that the design is not suitable.

2.4 GOAL 4: PROCESS: PROCESS IMPLEMENTATION EFFECTIVITY

The objective of implementation effectivity is to maximize the effectiveness of resources within the project scheduled activities. The attributes of this goal are:

- **Resource use** - The extent resource usage is appropriate for the phase of the project.

- **Completion Rates** - progress made in completing items such as peer reviews, or turnover of completed modules to CM.

2.4.2 PROCESS IMPLEMENTATION EFFECTIVITY ATTRIBUTES AND METRICS

The resource metric used by the SATC is personnel hours devoted to a set of life cycle activities. The measures collected are staff hours spent on a list of activities that is tailored for the project (and may have to change as the project progresses). Key activities that are measured include: requirements analysis, coding, testing, corrective action, training, and others.

Completion rates of tasks and product components are a good indicator of risk that a project will or will not be able to provide products on the needed schedule. Completion rates can be measured, and if a detailed schedule is available, completions can be compared to the schedule to provide planned versus actual charts. This, combined with the resource measures, gives a good picture of what is going on and the impact of the carryover of prior phase activities.

2.4.3 PROCESS IMPLEMENTATION EFFECTIVITY AND RISK

The risks that can be determined from the metrics of resource use are based on the appropriateness of the tasks to which resources are being applied at a given time during the life cycle. To the extent that those activities do not match the expected or planned activities, an element of risk may have been identified. For example, work on prior phase activities during the current phase is a risk indicator. If significant effort is being expended on requirements activities during the design phase (or worse, during the implementation phase), there is significant risk that the project will not be able to meet its schedule objective.

As an example of the use of completion rate data in determining risks, the completion rate of tests can be used to estimate the risk of completing all tests on time. The risk is measured by use of test report data. By determining the rate of completion of tests a simple calculation can show if the remaining tests can be completed in the time available.

3. DEVELOPING A METRICS PROGRAM

Once a developer decides to implement a metrics program, the next step is *How*. How a metrics program is developed can determine its success or failure. One approach is to investigate tools available for metrics collection, purchase the tool, then collect and attempt to apply whatever metrics are provided by the tool. This may work but has a major hurdle - what will the data collected tell the management and developers about their specific project. Data collected just because it is available has minimal value at best and usually ends up a waste of resources.

Successful metrics programs generally begin by focusing on a problem. At the start of the metrics program, goals must be established that addresses the problem. Related questions that management wants answered are identified then the data that is needed to answer these questions are specified. This leads to the tool specification for purchase or in-house development. Data collection can be expensive if not carefully monitored - the temptation is to collect all possible data and decide how to use it later.

The earlier benefits are seen by both management and developers, the sooner metrics programs are accepted. Metric programs should be designed to show visible benefits as soon as possible,

this is the key to continued support.

3.1 GOAL/QUESTION/METRIC (GQM) PARADIGM

The Goal/Question/Metric (GQM) Paradigm is a mechanism that provides a framework for developing a metrics program. It was developed as a mechanism for formalizing the tasks of characterization, planning, construction, analysis, learning and feedback. The GQM paradigm was developed for all types of studies, particularly studies concerned with improvement issues. The paradigm does not provide specific goals but rather a framework for stating goals and refining them into questions to provide a specification for the data needed to help achieve the goals. [l]

The GQM paradigm consists of three steps:

1. Generate a set of goals
2. Derive a set of questions
3. Develop a set of metrics

4. METRIC PROGRAM COSTS VS. BENEFITS

It is difficult to pin down the costs of a metrics program because metrics are usually just one aspect of an overall improvement program. When investigating the feasibility of starting a metrics program, it is often found that managers are individually collecting some form of data. This decreases initial program start-up cost. Accurate and complete measurements are not inexpensive; comprehensive metrics programs for software products and process annual costs can be 2 to 3 percent of the total software budget for collecting hard data. [3] Attempts to pin down the cost of metrics hide the real issue, however, developers don't really have a choice. The cost of not implementing a software metrics program can be measured in terms of project and business failures. Those projects and companies who make the investment in metrics have a competitive advantage over those who do not. They have the advantage of more informed and timely decisions that will ultimately make them more successful, with the best track records in terms of bringing software projects to completion and achieving high levels of user satisfaction. [2]

It is difficult, if not impossible, to place a dollar amount on the benefits of a metrics program because as in the case of risk management, you are trying to measure something that did not happen. The benefits derived are also not only applicable to the current project but to future projects. As with any new project, whether it is implementing a new engineering design or a metrics program, start up costs are high. But as management and staff become familiar with the tasks and tools are developed, the costs decrease to a low maintenance level.

5. CONCLUSION

The SATC applies goals to evaluate the quality of products (requirement documents through test applications) and provide risk information to project managers. Metric programs are initiated to

answer a question or provide numerical input to solve a problem.

The first step in developing a metrics program is to identify what are the goals or objectives of the program, then stay focused on them. The objectives can be expanded into specific goals using the structure of the Goal/Question/Metric templates.

The second step is to define the T attributes that are be measured. These attributes are a subset of the quality attributes and are chosen based on the project objectives and goals. If the GQM is used, some of the goals will relate to the attributes.

The third step in developing the metrics program is to clarify and quantify the goals. This is done by specifying questions and identifying metrics and data that are needed. At this point a tool is chosen based on the needs of the project.

The final and a very critical step is to close the loop - provide management with answers to their questions based on the metric analysis. The key to continued success of a metrics program is immediate, visible benefits. It must do the job it was designed to do and supply management with usable information to solve their current problem in a timely fashion.

REFERENCES

[1] Basili, V., and Rombach, H., "The TAME Project: Towards Improvement-Oriented Software Environments", University of Maryland, UMIACS-TR-88-8, 1/88.

[2] Grady, Robert, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.

[3] Hammer, T. et.al, "Requirement Metrics - Value Added", Requirements Engineering '97, 1/97.

[4] Hyatt, L. and Rosenberg, L., "A Software Quality Model and Metrics for Risk Assessment", *Journal for Software Quality*, 11/96.

[5] Rosenberg, L. and Hyatt, L., "Developing a Successful Metrics Program", STC '96, 4/96.

[6] Wilson, W. et.al, "Automated Quality Analysis of Natural Language Requirement Specification", ICSE97, 5/97.

Developing a Successful Metrics Program was Presented at the International Conference on Software Engineering, San Fransisco, CA, November 1997.